TUPLES

Tuples:

- Like list, Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Output:

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Sl. No.	Key	List	Tuple
1	**Type**	List is ***mutable***.	Tuple is ***immutable***.
2	**Iteration**	List iteration is ***slower*** and is time consuming.	Tuple iteration is ***faster***.
3	**Appropriate for**	List is useful for ***insertion*** and ***deletion*** operations.	Tuple is useful for ***read only*** operations like accessing elements.
4	**Memory Consumption**	List consumes **more memory**.	Tuples consumes **less memory**.
5	**Methods**	List provides many in-built **methods**.	Tuples have less in-built **methods**.
6	**Error prone**	List operations are more error prone.	Tuples operations are safe.

Similarities:

Although there are many **differences between list and tuple**, there are some similarities too, as follows:

- The two data structures are both sequence data types that store collections of items.
- Items of any data type can be stored in them.
- Items can be accessed by their index.

Difference in syntax

As mentioned in the introduction, the syntax for list and tuple are different.
For example:

```
list_num = [10, 20, 30, 40]
tup_num = (10, 20, 30, 40)
```

Allow Duplicates: Since tuples are indexed, they can have items with the same value:

Example:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Output:

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Tuple Length: To determine how many items a tuple has, use the len() function

Example:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Output:

```
3
```

Create Tuple With One Item: To create a tuple with only one item, you have to add a ***comma*** after the item, otherwise Python will not recognize it as a tuple.

Example:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Output:

```
<class 'tuple'>
<class 'str'>
```

Tuple Items - Data Types: Tuple items can be of any data type

Example:

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

print(tuple1)
print(tuple2)
print(tuple3)
```

Output:

```
('apple', 'banana', 'cherry')
(1, 5, 7, 9, 3)
(True, False, False)
```

Tuple Items - Data Types *(Continue)*: A tuple can contain different data types

Example:

```
tuple1 = ("abc", 34, True, 40, "male")
print(tuple1)
```

Output:

```
('abc', 34, True, 40, 'male')
```

type(): From Python's perspective, tuples are defined as objects with the data type 'tuple'

```
<class 'tuple'>
```

Example:

```
mytuple = ("apple", "banana", "cherry")
print(type(mytuple))
```

Output:

```
<class 'tuple'>
```

The tuple() Constructor: It is also possible to use the ***tuple()*** constructor to make a tuple.

Example:

```
thistuple = tuple(("apple", "banana", "cherry"))  # note the double round-brackets
print(thistuple)
```

Output:

```
('apple', 'banana', 'cherry')
```

Access Tuple Items: You can access tuple items by referring to the index number, inside square brackets

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Output:

```
banana
```

Negative Indexing: Negative indexing means start from the end. **-1** refers to the last item, **-2** refers to the second last item etc.

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Output:

```
cherry
```

When specifying a range, the return value will be a new tuple with the specified items.

Example:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Output:

```
('cherry', 'orange', 'kiwi')
```

***Note**: The search will start at index 2 (included) and end at index 5 (not included).*

Example:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

Output:

```
('apple', 'banana', 'cherry', 'orange')
```

Guess the output:

Example:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

Output:

?

Range of Negative Indexes: Specify negative indexes if you want to start the search from the end of the tuple

Example:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Output:

```
('orange', 'kiwi', 'melon')
```

Change Tuple Values: Once a tuple is created, you cannot change its values - ***unchangeable*** or ***immutable***.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Output:

```
("apple", "kiwi", "cherry")
```

Add Items:

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list:** Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)Output:

("apple", "kiwi", "cherry")
```

Output:

```
('apple', 'banana', 'cherry', 'orange')
```

Add Items*(continue)*:

2. **Add tuple to a tuple:** You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple

Example:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Output:

```
('apple', 'banana', 'cherry', 'orange')
```

Remove Items: Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items

Example:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

Output:

```
('banana', 'cherry')
```

Example:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Output:

```
NameError: name 'thistuple' is not defined
```

Unpacking a Tuple: When we create a tuple, we normally assign values to it. This is called "packing" a tuple

Example:

```
fruits = ("apple", "banana", "cherry")
```

Output:

```
('apple', 'banana', 'cherry')
```

But, in Python, we are also allowed to extract the values back into variables. This is called ***"unpacking"***:

Example:

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Output:

```
apple
banana
cherry
```

Unpacking a Tuple: When we create a tuple, we normally assign values to it. This is called "packing" a tuple

Example:

```
fruits = ("apple", "banana", "cherry")
```

Output:

```
('apple', 'banana', 'cherry')
```

But, in Python, we are also allowed to extract the values back into variables. This is called ***"unpacking"***:

Example:

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

Output:

```
apple
banana
cherry
```

Note: *The number of variables must match the number of values in the tuple, if not, you must use an* **** asterisk*** *to collect the remaining values as a list.*

Using Asterisk * : If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list

Example:

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red)
```

Output:

```
apple
banana
['cherry', 'strawberry', 'raspberry']
```

Using Asterisk * *(continue)*:

Example:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
print(red)
```

Output:

```
apple
['mango', 'papaya', 'pineapple']
cherry
```

Loop Through a Tuple: You can loop through the tuple items by using a for loop

Example:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Output:

```
apple
banana
cherry
```

*Note: We can also use **while** and **do-while** loop*

Join Two Tuples: To join two or more tuples you can use the **+** operator

Example:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

Output:

```
('a', 'b', 'c', 1, 2, 3)
```

Multiply Tuples: If you want to multiply the content of a tuple a given number of times, you can use the * operator

Example:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
print(mytuple)
```

Output:

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```